REMARKS

The Examiner is respectfully requested to enter the proposed amendment to claim 4 after final rejection to render the claim in better condition for allowance or appeal. The amendment corrects a typographical error and does not add new matter.

Claims 1-4, 11-14, and 21-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over US patent 5,297,150 (CLARK) in view of US patent 5,854,925 (SHIMOMURA).

Claims 5-10, 15-20 and 25-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over CLARK in view of SHIMOMURA and US patent 5,463,768 (CUDDIHY)

The Examiner is respectfully requested to withdraw the rejection of these claims in view of the following comments.

Claim 1

The method of claim 1 helps a programmer debug a program by telling the programmer which individual statements influenced a variable (the "error variable") that took on an undesired value during program execution, and by presenting those statements ordered in accordance with the probability they caused the error.

Assume, for example, that a program has three variables: X, Y and Z. After the program runs, a programmer sees that variable X took on a value of 1 at a point when the programmer wanted it to have a value of 0. The programmer therefore identifies variable X as an error variable and inputs the name X of the variable to the method recited in claim 1. The Examiner correctly observes that SHIMOMURA teaches that a programmer can identify an error variable. This is, of course, not new since programmers have been identifying error variables for as long as they have been writing programs and trying to debug them.

The applicant's method of claim 1 presents as output an "error set", the subset of all program statements that are relationally connected to the error variable. The notion of a statement being "relationally connected" to a particular variable means that the statement is directly or indirectly responsible for setting the value of that particular variable. Thus, while the <u>input</u> to the applicant's method is an error variable identification, the <u>output</u> of the applicant's method of claim 1 is a subset of all statements that are directly or indirectly responsible for setting the value of the error variable during program execution. See paragraph 27 of the application for a description of the error set.

The applicant respectfully disagrees with the Examiner's statement that

> "CLARK disclosed flow graphs composed of statements which are relationally connected (the nature of connecting nodes of the graph) to the error disclosed by Shimomura,"

A flow graph is a graphical depiction of the order(s) in which program statements can be executed when the program includes conditional branching statements such as, for example, an IF statement. As illustrated in the flow graph CLARK's FIG. 2, each numbered node of the flow graph represents a block of statements ending with a conditional branching statement (col. 4, lines 50-51). The lines connecting the nodes of the graph indicate separate possible directions of program flow between blocks of statements, depending on the outcomes of the conditional statements at the end of the statement blocks (col. 4, lines 20-29). For example, the lines connecting node 2 of CLARK's FIG. 2 to nodes 12, 3 and 4 show that, depending on the outcome of the conditional statement at the end of the block of statements represented by node 2, the next statement to be executed could be the first statement of block 12, the first statement of block 3, or the first statement of block 4. Thus, the lines connecting the nodes of the flow graph of FIG. 2 indicate a <u>program flow relationship between blocks of code statements</u>, however they do not indicate that any particular statement is <u>relationally connected to some particular error value</u> as recited in claim 1. CLARK's flow graph does not tell us which statements of a program are directly or indirectly responsible for setting the value of any particular variable.

CLARK describes a method for identifying flow paths in a program, not statements that are relationally connected to a particular variable. A "flow path" is a possible sequence in which statements can be executed. For example, one flow path of CLARK's FIG. 2 includes the set of statement blocks {1, 2, 3} since when the program runs, it is possible that it will execute only statement blocks 1, 2 and 3. Another flow path includes the set of statement blocks {1, 2, 12, 8, 9, 10, 11, 2, 3}. A program with conditional branching statements can have many flow paths.

In addition to identifying each possible flow path within a program, CLARK also assigns a weight to each flow path to indicate the complexity of the flow path. The flow paths are then presented to a programmer in order of complexity. CLARK teaches that a programmer ought to test the most complex ("critical") flow paths first and most thoroughly (col. 6, lines 41-46), because these are the paths most likely to contain programming errors.

Thus, the following are true:

1. CLARK teaches a system that produces as output several subsets of statements (flow paths) that are included in a program,
2. CLARK's system weights or ranks each subset according to complexity,
3. CLARK teaches that likelihood that a flow path contains a programming error increases with the complexity of the flow path.

Another aspect of the method described by claim 1 is the notion of assigning a priority value to each program code statement in the error set indicating a computed probability that that particular code statement is the source of the error in the error variable. The applicant respectfully disagrees with the Examiner's statement that

> "CLARK provided weighting to indicate likelihood of error. Even if complexity is the underlying factor in CLARK, it still demonstrates a system of probability in failure. Failure stemming from the error provided by Shimomura."

The applicant's claim 1 does not recite weighting or ranking flow paths according to complexity. It recites determining and error set (the subset of statements that are relationally connected to a particular variable that was shown to have taken on an incorrect value during program execution) and then weighting or ranking each individual program statements of the error set according to the probability that that particular statement caused the error in the error variable. Thus the claimed priority/weighting/ranking system gives weight to relationally connected program statements, whereas CLARK teaches giving weight to program flows.

The applicant's method tells a programmer which <u>individual statement</u> of all the statements in a program that could have affected the value of a variable was most likely to have caused an error in the value of that variable. CLARK's method can only tell the programmer which program flow is most complex. While the most complex program flow may be the most likely to include programming errors, it is not necessarily true that the most complex program flow caused an error in the value of a particular variable. It may well be true that the most complex program flow was not the program flow that was executed during program execution and that none of the statements in the most complex program flow had any effect of the error variable because they were not executed. Even if the most complex program flow was executed, CLARK's method does not give the programmer any idea of the relative likelihood that individual statements of that program flow caused the error, as does the method of the applicant's claim 1. A program flow has many statements, and it may well be that most of the statements in the most complex program flow have no effect whatsoever on the value of an error variable. Even if every statement in the most complex program flow can directly or indirectly affect the value of the error variable, CLARK's weighting system doesn't tell the programmer which statement would be most likely to have caused the error, since the weight applies to the entire program flow.

Assume a programmer runs a program and finds that the program sets some variable X to an unexpected value, and the programmer wants to know why that happened. He wants to know which individual statement caused the error, and he would like a hint as to which statements could have caused it. He supplies the name of variable X as input to the method of claim 1 and the method gives him the error set, the subset of all statements that directly or indirectly affect the value of variable X, with the statements being ordered to reflect the probability that the statements caused the error. That is very helpful information to the programmer, because he can immediately direct his attention to the particular individual statements that were most likely to have caused variable X to take on the unexpected value.

Suppose instead that the programmer chooses to employ CLARK's method for analyzing a program that produces not an error set, but a list of program flows, where each program flow is a different sequence of statements that could be executed when the program is run. First note that even though the programmer knows the identify of the error variable X, and even though programmer may be intimately familiar with the teachings of SHIMOMURA, the programmer would not be motivated to supply the name of variable X as input to CLARK's method because CLARK's method does not require an input identifying an error variable and is not adapted to receive or process such input, or to produce an output that is in any way influenced by such an input. The only input to CLARK's method is a program listing. That is the only input it needs because it all it does is identify program flows and rank them by complexity. CLARK's method does not care what variables the program statements may or may not influence when it categorizes statements according to program flow.

Note also that the output of CLARK's method doesn't tell the programmer which program flow was actually executed, so the programmer doesn't know which program flow to look at in order to determine why variable X took on an unexpected value. It would not be particularly helpful for the programmer to investigate the most complex program flow when that program flow was not executed. The statement that caused the error might not even be included in the most complex program flow.

Even if the programmer can somehow determine which of the program flows was executed, the output of CLARK's method does not tell him which <u>individual statements</u> of that program flow are relationally connected to the error variable.

Finally, even if the programmer could determine which of the program statements of the program flow that was executed were relationally connected to the error variable, CLARK's method doesn't provide the programmer with any indication as to the probability that each such statement actually caused the error.

Thus, the output of CLARK's method would not be very helpful to a programmer who wants to know why variable X took on an unexpected value. But of course, CLARK's method is not intended to help a programmer determine such a thing, and CLARK does not suggest that it does. CLARK's method is intended only to point out which of the possible program flows are most complex. This is helpful information for a programmer who, for example, wants to simplify a program, but it is not very useful to a programmer who wants to know why some particular variable took on an unexpected value.

Claim 1 is therefore patentable over the combination of CLARK and SHIMOMURA because neither CLARK nor SHIMOMURA teach or suggest a method for analyzing program code to determine the subset of statements that are directly or indirectly responsible for the value of an error variable (the error set), and for ordering the statements when presenting the error set to reflect the probability that each individual statement of the error set caused the error. Also it would not be obvious to provide the error variable indication input taught by SHIMOMURA is input to the method taught by CLARK because CLARK does not teach or suggest a method adapted to receive and process such an input or to produce an output that depends on such information.

Claim 2

Claim 2 depends on claim 1 and is patentable over the combination of SHIMOMURA and CLARK for similar reasons. Parent claim 1 recites that the error set is a subset of individual code statements, which are relationally connected to a variable that had an error in value during program execution. Claim 2 recites that the statements in the error set are presented in an order based on likelihood of each statement being a source of the error. CLARK's method presents a set of flow paths as output, rather than an error set of individual statements and it is the <u>flow paths</u>, not <u>individual statements</u>, that CLARK provides as output ordered to reflect complexity weighting.

Thus while the output of CLARK's method indicates the various flow paths of a program, and indicates the relative complexity of the various flow paths, it doesn't indicate which individual statements in a program had an influence of the value of a particular variable. ALSO, CLARK's method does not indicate the relative probability that each such statement could have been the cause of the error in the variable value as recited in claim 2.

Claim 3

Claim 3 depends on claim 1 and is patentable over the combination of SHIMOMURA and CLARK for similar reasons. Claim 3 further recites that each program is presented with the associated priority value. The priority value, as recited in claim 1, indicates a computed probability that the associated program statement is the source of an error in the value of a particular variable that occurred during program execution. CLARK assigns a weight to each program flow (comprising more than one statement) indicating the complexity of the program flow. It is program flows, and their associated complexity weighting, that CLARK's system presents as output (col. 6, line 33-46.). Thus CLARK's system presents program flows with associated complexity weighting rather that a set of statements relationally connected to an error variable along with a priority value for each individual statement indicating a computed probability that that particular statement caused a particular error in a particular program variable during program execution, as recited in claim 3.

Claim 4

Claim 4 depends on claim 1 and is patentable over the combination of SHIMOMURA and CLARK for similar reasons. Claim 4 further recites

> "obtaining an error cycle that is an execution cycle in which the error variable obtains the error value."

In many cases, program execution is organized into a set of successive execution cycles wherein a program is executed from start to end during each execution cycle, with data output of one cycle being used as data input to a next execution cycle. For example an HDL program that simulates behavior of a circuit that is clocked by a clock signal, undergoes one execution cycle for each "tick" (edge) of the clock signal, and during each execution cycle computes how the states of all of the circuit outputs would change in response to the clock signal edge. See specification paragraph 15.

Claim 4 recites obtaining an execution cycle in which an error value took on the undesired "error value". CLARK does not discuss such a step. SHIMOMURA teaches that a user provides an indication as to the location within a program listing of a statement that caused a variable to take on an unexpected value. See SHIMOMURA col. 2, lines 40-43. However, providing an indication as to the location or position of a statement within a listing is not the same thing as providing an indication as to an execution cycle in which the statement caused the error in the value of a variable. A single statement can be executed in thousands of different execution cycles but may cause the error during only one of the program execution cycles.

Neither CLARK nor SHIMOMURA discuss the notion of program execution cycles. The Examiner states that the "prior experience" discussed by CLARK col. 6, lines 10-12 relates to the notion of program execution cycles. These lines of CLARK are quoted below:

"Next the nodal weights are revised either as a result of prior experience, (e.g. the path has been previously searched) or as a result of a change in heuristic that indicates a revised value is to be assigned."

That statement was made in the context of explaining a method for analyzing a program listing to identify flow paths and assign a weight to each flow path based on complexity. The "prior experience" CLARK is talking about relates to a prior experience of having previously searched a flow path within a program code listing to determine its complexity. It has nothing at all to do with program execution cycles as the term is used in the applicant's specification and claims. CLARK is not concerned with program execution cycles because CLARK's method does not require a program to be executed before it is used. CLARK's method is only designed to identify program flows and determine how complex they are by studying a program listing, and does not require as input any information obtained by actually executing the program. It has no use for such information. The applicant's method as recited in the claims does require a program to be executed before implementing the method because it is necessary to execute a program in order to identify an error variable - which is a variable that took on an undesired value <u>during program execution</u>.

Claim 4 further recites:

"obtaining an execution set comprising all program statements that are executed in the error cycle".

Neither CLARK nor SHIMOMURA teach this step. CLARK is concerned with determining the various alternative program flows that could be executed, not the set of program statements that actually was executed.

Claim 4 further recites:

"utilizing the program code statements in the execution set to generate the error set, every program code statement in the error set being in the execution set and also being relationally connected to the error variable".

Neither CLARK nor SHIMOMURA generate an error set of statements selected from among the set of statements executed during a particular execution cycle which are relationally connected to an error variable.

Claim 5

Claim 5 further recites the program code includes a "correct variable having a value that agrees with a second desired value in the error cycle" and "obtaining a first sensitized set for the correct variable...comprising at least a program code statement...relationally connected to the correct variable, that is excluded in the first execution cycle and that is in the error set". In other words, claim 5 recites determining a set of program statements appearing in the error set that previously produced correct values for a variable. Since these statements are less likely to cause the error in the error variable, claim 5 further recites a step of "applying a scaling function to the priority value associated with the identical program code statement in the error set... setting a reduced computed probability that the associated program code statement...is the error source of the error variable." The Examiner correctly points out that CLARK and SHIMOMURA fail to teach this but incorrectly points to CUDDIHY as teaching this.

CUDDIHY relates to a method for analyzing "historical error logs" of the type shown in CUDDIHY's FIG. 2 generated by equipment which include an error code, time of error and various comments. It is not concerned with analyzing program statements, but it could be used to analyze error codes produced by an executing program. CUDDIHY's abstract indicates that the error logs are analyzed to find common sections of an error log that are similar, labelling common sections as "blocks", weighing each block according to its value in diagnosing an equipment fault, and then using the block and weighting information in the future to help diagnose equipment faults.

Thus CUDDIHY cannot be cited for producing a "first sensitized set...comprising at least a program code statement … relationally connected to the correct variable" as recited in claim 5, because CUDDIHY processes error codes that could be produced by a program but not the code statements forming the program itself. It groups error codes, not program statements. It assigns weights to blocks of error codes, not to individual program statements.

In the previous office action, the Examiner cited CUDDIHY Table 1 as teaching the "correct variables" recited in the applicant's claim 5. The Examiner suggests that the correct variables appear in blocks of the historical error logs. CUDDIHY's Table 1 (col. 6, lines 23-33) merely lists a set of blocks and shows the weight assigned to each block as being an inverse function of the number of occurrences of the block in an error log. The recited "correct variable" is a program variable that has a correct value during program execution. The blocks of CUDDIHY's Table 1 refer to sections of an error log as illustrated for example in CUDDIHY's FIG. 2. Nothing in the error log indicates anything about variables having correct values. Error logs do not keep track of variables that have correct values. They keep track of events that indicate device failures. Thus, the Examiner's assertion that blocks of a historical error log represent the applicant's "correct variables" is incorrect.

In the current office action the Examiner states:

> "Further correct values as demonstrated in the previous office action are shown through historical sensitized set values in the table 1. As originally interpreted the past occurrences produce 'correct' values. A constant of zero is demonstrated in column 5, lines 33-34 of CUDDIHY."

The "zero" discussed in this section of CUDDIHY relates to a zero weight assigned to a block of error log messages and has nothing at all to do with the "correct variables" recited in the applicant's claim 5 which relate to variables having values that a program sets.

It is not entirely clear as to what the Examiner means by the following statement and it is not clear as to which of the applicant's arguments or claims the statement was intended to apply:

> "Seventh, once again revision (a repetitive process of assigning the values requiring multiple cycles for processing) is clear from the cited passage of CLARK".

The applicant assumes, however, that the statement is intended relate to claim 5 and to mean the CLARK's discussion of "prior experience" implies that CLARK's method is somehow dealing with repetitive program execution cycles. While CLARK's method may repetitively process a program listing to identify and determining the complexity of flow paths, it has nothing to do with analyzing a program that has undergone several program execution cycles to determine which particular program statements were likely to have caused an error in a variable value during one particular program execution cycle. CLARK's method has nothing to do with an error set, with an error value, with determining a set of program statements appearing in an error set that previously produced correct values for a variable, with "applying a scaling function to the priority value associated with the identical program code statement in the error set", or with "setting a reduced computed probability that the associated program code statement...is the error source of the error variable" as recited in the applicant's claim 5.

Claims 6-9

The Examiner does not appear to have provided any specific rebuttal to the applicant's arguments relative to the additional limitations of claims 6-9 provided in response to the previous office action. The applicant stands by those arguments.

Claims 10-30

Claims 10-30 are patentable over the combination CLARK, SHIMOMURA and CUDDIHY for reasons cited above in connection with their parent claims.

In view of the foregoing amendments and remarks, it is believed that application is in condition for allowance. Notice of Allowance is therefore respectfully requested.

Respectfully submitted,

[illegible]

Daniel J. Bedell
Reg. No. 30,156

SMITH-HILL & BEDELL, P.C.
16100 N.W. Cornell Road, Suite 220
Beaverton, Oregon 97006

Tel. (503) 574-3100
Fax (503) 574-3197
Docket: SPRI 3135

Certificate of Facsimile Transmission

I hereby certify that this paper is being facsimile transmitted to the Patent and Trademark Office on the date shown below.

Penelope Stockwell

Penelope Stockwell

June 17, 2005

Date